UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Automatización de test regresivos de interfaz de usuario para aplicaciones C#.NET

Autor/es

AMAIA URRA MARTÍNEZ

Director/es

ARTURO JAIME ELIZONDO

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2019-20

Automatización de test regresivos
de interfaz de usuario para aplicaciones C#.NET
, de AMAIA URRA MARTÍNEZ

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Automatización de test regresivos de interfaz de usuario para aplicaciones C#.NET

Realizado por:

Amaia Urra Martínez

Tutelado por:

Arturo Jaime Elizondo

Logroño, Septiembre, 2020

Resumen

El objetivo de este proyecto es analizar una herramienta para la realización de tests regresivos automatizados, así como diseñar, ejecutar y mantener un conjunto de pruebas automáticas sobre una aplicación real en desarrollo. Las herramientas a utilizar son VQ Light, desarrollada por la propia empresa donde se realiza el proyecto y Selenium, una interfaz de programación de aplicaciones (API) capaz de emular las acciones que realizaría un usuario humano a través de su teclado y ratón, permitiendo por tanto la interacción del código con un navegador.

Este TFG parte de la necesidad de asegurar la calidad de la aplicación en un entorno ágil, donde los despliegues a un entorno de producción son frecuentes. Si las pruebas sobre la interfaz del usuario se realizan de modo manual sólo cabe recortar la calidad o ampliar el tiempo entre los despliegues, como se venía haciendo hasta el momento. Esta memoria trata de explicar el proceso seguido y las principales lecciones aprendidas durante la realización de test regresivos automatizados sobre la interfaz de usuario.

Abstract

The objective of this project is to analyze a tool for the realization of automated regression testing, as well as to design, execute and maintain a set of automatic tests on a real application under development. The tools to be used are VQ Light, developed by the company where the project is carried out and Selenium, an application programming interface (API) capable of emulating the actions that a human user would perform through their keyboard and mouse, thus allowing the code to interact with a browser.

This TFG is based on the need to ensure the quality of the application in an agile environment, where deployments to a production environment are frequent. If the tests on the user interface are performed manually, it is only possible to cut the quality or extend the time between deployments, as it has been done up to now. This report tries to explain the process followed and the main lessons learned during the automated regression testing on the user interface.

Índice

1. Introducción

En este apartado justificamos la elección del proyecto, los motivos que llevaron a realizarlo, la situación por la que se decidió poner en marcha este proyecto y los objetivos principales.

1.1. Contexto

El presente TFG (trabajo de fin de grado) se encuadra en un proyecto desarrollado en la empresa *Cotecna Inspección S.L.*, sección española de *Cotecna Inspection S.A.* sita en Suiza. La empresa me contrató como probadora de software (tester) y tras un periodo de dos años en el puesto he podido experimentar el gran esfuerzo que conlleva la realización de pruebas de regresión. Así que decidí formarme en la automatización de pruebas con objeto de simplificar y mitigar esta tarea.

En este TFG se analizará un software concreto de automatización de pruebas denominado ValueQuest Light®. Lo que pretendemos es automatizar las pruebas de la interfaz gráfica de usuario emulando su actividad. El objetivo de este TFG es identificar qué tipos de pruebas de regresión merece la pena automatizar. Para ello se ha utilizado la herramienta para automatizar pruebas en un proyecto real, de modo que se puedan extraer lecciones aprendidas y conclusiones.

Las *pruebas de regresión* son todas aquellas que cubren la funcionalidad (a alto nivel) de la aplicación en desarrollo. Deben realizarse antes de cada despliegue en un entorno de *pruebas de aceptación de usuario*, conocido como entorno UAT *(User Acceptance Testing)*.

En Cotecna, se utilizan metodologías ágiles y se intenta realizar una integración contínua, siempre que sea posible. Esto supone que se realizan despliegues en entornos UAT siempre que se introducen nuevas funcionalidades o correcciones, por pequeñas que estas sean. También se realizan antes de pasar el software desarrollado a producción. Esto significa que las pruebas de regresión, sobre un software concreto en desarrollo, se repiten con mucha frecuencia.

Crear y mantener pruebas automáticas tiene un coste elevado. Por ello, hay que elegirlas bien y priorizarlas según su importancia, para lograr la estabilidad de la aplicación en desarrollo. Como se ha mencionado, las pruebas de regresión se ejecutan con mucha frecuencia, y por ello merece la pena invertir tiempo en hacer una buena selección.

Para implementar la automatización se va a utilizar el entorno de pruebas de software Selenium WebDriver. Selenium permite manejar el controlador de un navegador, emulando el comportamiento de un usuario sobre una interfaz gráfica web. Mediante una librería específica, se pueden utilizar varios lenguajes de programación, que en nuestro caso será C#.

1.2. Motivación

Hay tres aspectos destacables que motivaron la elección de este tema para realizar el TFG:

1. **Mejorar la formación**. Antes de realizar este TFG hacía exclusivamente testing manual. La elaboración automática de pruebas supone la fusión de los roles de programador y

probador-tester. Al preparar el certificado ISTQB Foundation (lo obtuve hace tres años) estudié las nociones básicas de automatización de pruebas. Pero, para completar esta formación, faltaba ponerlo en práctica con herramientas profesionales apropiadas, lo cual resultaba un reto profesional interesante.

2. **Mejorar la productividad.** Crear buenos conjuntos de pruebas de forma automatizada ahorra tiempo. Así, el probador-tester se libera antes de la tarea y se puede dedicar a realizar otras pruebas.
3. **Mejorar el perfil profesional.** El perfil de probador-tester de automatización de pruebas es uno de los más demandados en el área de Barcelona, ciudad donde resido actualmente. La remuneración media de este perfil es un 20% superior al de los probadores-tester manuales. Por tanto la realización del TFG contribuye a mejorar este perfil profesional.

2. Antecedentes

Para poder entender el punto de partida, empezaremos explicando qué tipo de software se realiza en la empresa donde se encuadra el TFG y cuál es el método de desarrollo que se utiliza. Generalmente se trabaja en la gestión de operaciones de aduanas y los clientes son gobiernos de países. La empresa no vende software de forma aislada, sino que lo incluye como parte de su operativa de gestión de aduanas. Algunos empleados se dedican a inspeccionar físicamente las importaciones y exportaciones. Estos inspectores comprueban que los productos corresponden en cantidad, forma y precio a lo declarado en la aduana. Valuequest Light® recoge los resultados de estas inspecciones, a las que llamamos *Evidencias*. Esto quiere decir, que la mayor parte de los usuarios de nuestro software son internos de la empresa.

Para el desarrollo de software se utiliza la conocida metodología SCRUM. Con SCRUM se agiliza la entrega al cliente de sucesivas versiones del software que funcionan, aportando cierto valor sobre la versión anterior. El desarrollo de cada versión se llama iteración y se hace en un plazo de tiempo bastante breve. Este proceso iterativo comienza con la entrega de un producto viable mínimo o MVP (*minimum viable product*) y se van construyendo sobre él continuas mejoras según las necesidades manifestadas por el usuario. Este planteamiento afecta a la división del trabajo, los plazos de entrega de los subproductos y el proceso de desarrollo. El ideal es conseguir la integración continua. Además, SCRUM ayuda a que el equipo sea consciente de las tareas a realizar y de los plazos de entrega.

Desde el punto de vista de la prueba del software, esta forma de trabajar tiene un inconveniente. Cada versión desplegada en los servidores de producción debe ser estable, es decir sin errores. Por tanto, en cada despliegue, los responsables de probar el software, conocidos como QA (*Quality Assurances*), debemos probar toda la aplicación, asegurando que el producto tiene la calidad adecuada. Es decir, tenemos que garantizar que al corregir o mejorar el código no se han incluido defectos. Esto lo conseguimos realizando los llamados *tests de regresión*.

ValueQuest® es una aplicación web desarrollada en nuestra empresa con Silverlight (un plugin de Microsoft para el *framework* .Net especializado en interfaces web). Una de las características menos amables de Silverlight, es que no permite automatizar las pruebas. Esto significa que, en cada despliegue de la aplicación en el entorno previo a producción (UAT o entorno de *pruebas de aceptación de usuario*), los QA debemos invertir casi todo nuestro

tiempo y esfuerzo haciendo las pruebas de regresión manualmente. Solemos tardar 15 horas de media para hacer la regresión de una versión completa de la aplicación. Los tests automáticos cuestan menos de 1 hora. Hay que tener en cuenta que al seguir la metodología SCRUM esta actividad se realiza cada poco tiempo.

Como Silverligth no permitía automatizar pruebas y exigía utilizar Internet Explorer, nuestra empresa decidió reimplementar ValueQuest® sin Silverligth. Esta nueva versión se llama ValueQuest Light®. El *frontend* (interfaz) se desarrolló en Angular y Typescript, y el *backend* (lógica de negocio) en C#. Los test regresivos formarían parte de mi responsabilidad como QA. Estos test pueden ser funcionales, de interfaz de usuario, de API (interfaz de programación de aplicaciones) o no funcionales.

En el caso de las API, la interfaz de programación constituye una capa de abstracción entre el código del *backend* y los resultados presentados en el *frontend* de la aplicación. Los QA utilizamos dicha abstracción para identificar errores del *backend* sin confundirlos con errores de presentación en el *frontend*. En la empresa las pruebas sobre la API se hacen de forma manual escribiendo código javascript y ejecutándolo mediante la herramienta Postman. Estas pruebas también se podrían automatizar. Las llamadas realizadas desde el código javascript pueden ser de tipo POST, GET, PUT o DELETE equivalentes a CRUD (Create, Read, Update, Delete).

La empresa únicamente hace pruebas no funcionales para probar el rendimiento. Se utiliza la herramienta LoadRunner, que emula el comportamiento de 50 usuarios (o más en su versión premium), accediendo y explotando la aplicación de forma simultánea. Se utiliza la versión de prueba ya que las aplicaciones de la empresa rara vez superan los veinte usuarios simultáneos.

En este TFG nos hemos centrado en las pruebas automáticas de Interfaz de usuario, utilizando Selenium y ValueQuest Light®, como ya hemos explicado.

No habrá interesados de la empresa que participen como tales en el TFG, habiéndose asumido como iniciativa personal. Sin embargo, los resultados del análisis pueden resultar de interés para la empresa, en particular para el personal dedicado a la prueba de software.

3. Alcance y objetivos

El objetivo del TFG es el análisis de una herramienta (Selenium) que permita la automatización del test de interfaces, adaptarlo al caso particular de una aplicación concreta (ValueQuest Light®)y sacar conclusiones y lecciones aprendidas sobre su utilidad práctica en este tipo de desarrollos. Lo que interesa para nuestro objetivo es utilizar una librería que permita programar la interacción con una interfaz web. Es decir, que se trata de librerías que nos permiten emular las acciones realizadas por los usuarios sobre la interfaz (con su ratón y su teclado) y reflejarlas en líneas de código. De esta forma se puede realizar el mismo test regresivo sobre la interfaz tantas veces como sea necesario, lo que en nuestro caso se traduce en realizarlo una vez por cada nueva versión de la aplicación. Tras una valoración informal de varias de estas librerías se seleccionó Selenium, que se puede utilizar con C# y es de código abierto.

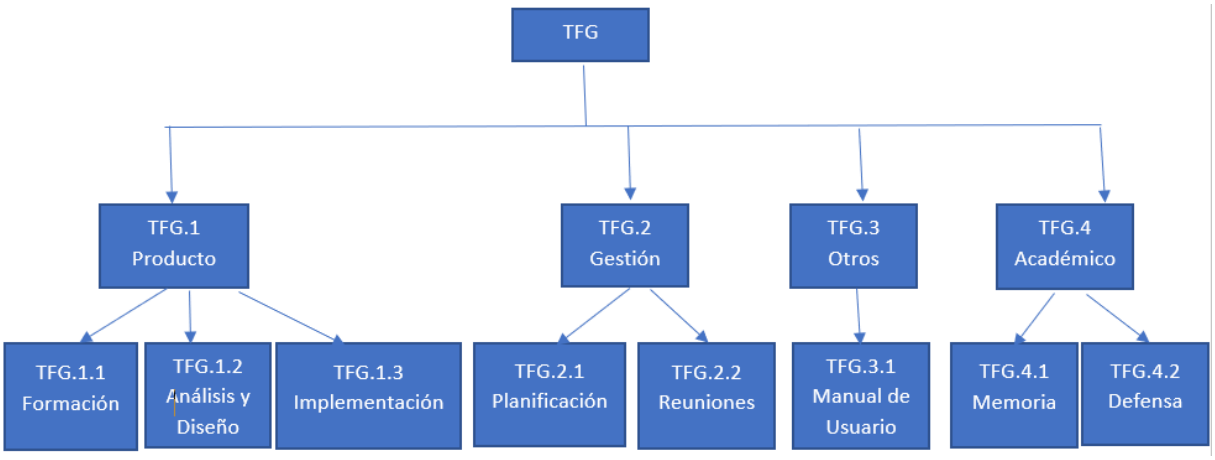

Figura 1. EDT del proyecto.

4. Gestión

La planificación de este proyecto ha sido simple ya que se el proyecto ha explorado tecnologías y técnicas que se ponían en marcha por primera vez en la empresa. El objetivo de la gestión ha sido fundamentalmente la extracción de datos interesantes que sirvan como criterio de estimación de futuros proyectos en el área.

4.1. Alcance (requisitos)

En un equipo que trabaja con metodologías ágiles, el trabajo de mejora se alarga en el tiempo desde que se entrega una primera versión del producto, por ello decidí acotar el alcance de este proyecto al primer MVP (producto viable mínimo) de la migración de nuestro producto, VQ Light.

En la automatización de este MVP se incluye la codificación de los casos de prueba regresivos de la consulta e inserción de evidencias de transacciones de importación y exportación así como un manual básico de usuario de la aplicación a testear para facilitar una mejor comprensión de los tests creados.

4.2. Cronograma

En la Figura 2 se presentan los principales hitos considerados y las Figuras 3 y 4 contienen el cronograma con los periodos de realización correspondientes a los paquetes de trabajo de la EDT de la figura 1.

Hito	Febrero				Marzo				Abril				Mayo					Junio				Julio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
Inicio TFG	◆																						
Depósito																					◆		
Académicos																							
Defensa TFG																							◆
Memoria revisar																			◆				

Figura 2. Diagrama de hitos principales del proyecto.

Periodo ejecución paquete TFG	Dic.		Enero		Febrero				Marzo				Abril				Mayo					Junio	
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23
1.1. Formación																							
1.2. Análisis y Diseño																							
1.3. Implementación																							
2.1. Planificación																							
2.3. Reuniones																							
3.1. Manual de usuario																							
4.1. Memoria																							

Figura 3. Cronograma del proyecto, primera parte.

Paquete TFG	Junio		Julio				Agosto				Septiembre			
	S24	S25	S26	S27	S28	S29	S30	S31	S32	S32	S33	S34	S35	
4.1. Memoria														
4.2. Defensa														

Figura 4. Cronograma del proyecto, segunda parte.

4.3. Recursos (dedicaciones)

Las Figuras 4 y 5 contienen la dedicación horaria a los diferentes paquetes de trabajo a lo largo de las semanas.

Dedicación horas paquete TFG	Dic.		Enero		Febrero				Marzo				Abril				Mayo					Junio		Total
	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	
1.1. Formación	4	4	4	4	4	4	4	4	4	4	4													44
1.2. Análisis y Diseño				8	8	6	6	6	6	4	4													48
1.3. Implementación				4	10	12	12	12	12	6	8	8	8	8	8	8	8	8	8	8	8	6		160
2.1. Planificación	3	3	3																					9
2.2. Reuniones					2						2									2				6
3.1. Manual de usuario																	4	4	4	4				12
4.1. Memoria																					4	4	4	

Figura 4. Horas de dedicación a cada paquete por semana redondeadas a horas enteras. primera parte

Dedicación en horas paquete TFG	Junio		Julio				Agosto				Septiem.			Total
	S24	S25	S26	S27	S28	S29	S30	S31	S32	S32	S33	S34	S35	
4.1. Memoria	6	6	6	6	6	6	6				4	4		62
4.2. Defensa													1	1
TOTAL														342

Figura 5. Horas de dedicación a cada paquete por semana redondeadas a horas enteras. segunda parte

5. Análisis y diseño

5.1. Qué casos de prueba se han automatizado

El testing automático tiene un alto coste para las empresas. Por un lado está el tiempo de codificación de las pruebas, y por tanto el coste asociado a su desarrollo, que suele superar al tiempo de ejecución manual de pruebas equivalentes. Por otro lado está el coste de mantenimiento del conjunto de pruebas, ya que es necesario revisarlas cada vez que se introducen cambios en la aplicación a testear. Por tanto, es muy importante identificar cuándo merece la pena automatizar las pruebas y cuándo no.

En este trabajo se han considerado los siguientes dos **criterios** para decidir qué pruebas se iban a automatizar:

1. Criticidad: El objetivo de la automatización de pruebas regresivas es garantizar la estabilidad de la aplicación. Por ello se priorizará este tipo de pruebas en los módulos más importantes desde el punto de vista de los usuarios y cuyo funcionamiento erróneo pueda suponer mayor perjuicio para el prestigio de la compañía.
2. Coste vs beneficio: En algunas ocasiones el coste de crear y mantener ciertas pruebas supera al perjuicio que pueda causar el defecto en el entorno de producción. En otras ocasiones estas pruebas pueden ser sustituidas por un test exploratorio sencillo. En esos casos, es preferible no diseñar ni codificar el test y hacerlo de forma manual.

Considerando los criterios anteriores, se han creado test regresivos automatizados para el siguiente **conjunto de pruebas**:

1. Añadir una evidencia:
    1. Iniciar sesión con un usuario administrador
    2. Hacer clic en el botón + ADD EVIDENCE
    3. Rellenar los datos del formulario
    4. Hacer click en el botón SAVE
    5. Hacer una búsqueda utilizando el valor añadido en el campo “Product Description”
    6. Comprobar que la búsqueda obtiene resultados
2. Realizar una búsqueda:
    1. Iniciar sesión con un usuario administrador
    2. Hacer clic en “Advanced Filters” para que los filtros avanzados sean visibles.
    3. Rellenar los filtros de búsqueda “Evidence Type”, “Importer”, “Goods Description”, “Unit” y “Origin Country” con datos de una evidencia existente.
    4. Hacer click en el botón SEARCH
    5. Comprobar que los resultados tienen los valores de los campos introducidos en los filtros de búsqueda
3. Reiniciar los filtros de búsqueda:
    1. Iniciar sesión con un usuario administrador
    2. Hacer clic en “Advanced Filters” para que los filtros avanzados sean visibles.
    3. Rellenar los filtros de búsqueda “Evidence Type”, “Importer”, “Goods Description”, “Unit” y “Origin Country” con datos de una evidencia existente.

4. Hacer click en el botón RESET
5. Comprobar que todos los valores de los filtros de búsqueda están en blanco

4. Realizar una búsqueda con filtros avanzados:
    1. Iniciar sesión con un usuario administrador
    2. Hacer clic en “Advanced Filters” para que los filtros avanzados sean visibles.
    3. Rellenar los filtros de búsqueda “Evidence Type”, “Importer”, “Goods Description”, “Unit” y “Origin Country” con datos de una evidencia existente.
    4. Hacer click en el botón SEARCH
    5. Comprobar que todos los valores de los filtros de búsqueda están en blanco.
5. Abrir detalle de una evidencia:
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Guardar el valor “File Reference” del primer resultado
    4. Hacer click en el botón del primer resultado
    5. Comprobar que el valor del campo “File Reference” coincide con el ”File Reference de la evidencia abierta.
6. Exportar a un fichero Excel evidencias (500 por defecto):
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Hacer click en el botón
    4. Comprobar que se descarga un excel con las primeras 500 evidencias del resultado de la búsqueda.
7. Exportar a un fichero Excel un número específico de evidencias:
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Introducir el valor “20” en el filtro “Number of items to export”
    4. Hacer click en el botón
    5. Comprobar que se descarga un excel con las primeras 20 evidencias del resultado de la búsqueda.
8. Intentar exportar evidencias a un fichero Excel sin resultado de búsqueda:
    1. Iniciar sesión con un usuario administrador
    2. Rellenar los filtros de búsqueda con caracteres aleatorios.
    3. Hacer click en el botón SEARCH
    4. Hacer click en el botón
    5. Comprobar que no se descarga ningún archivo y aparece el siguiente mensaje de error: “ERROR: Search result missing”.
9. Intentar exportar un número de evidencias no soportado (>500):
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Introducir el valor “501” en el filtro “Number of items to export”
    4. Hacer click en el botón
    5. Comprobar que no se descarga ningún archivo y aparece el siguiente mensaje de error: “ERROR: The limit is 500”.
10. Intentar exportar un número de evidencias erróneo (negativo):
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Introducir el valor “-1” en el filtro “Number of items to export”
    4. Comprobar que el símbolo ‘-’ no aparece en el filtro

5. Hacer click en el botón
6. Comprobar que se descarga un archivo excel con la primera evidencia del resultado de búsqueda

11. Exportar a PDF el detalle de evidencia:
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Guardar el valor "File Reference" del primer resultado
    4. Hacer click en el botón del primer resultado
    5. Hacer click en el botón de la ventana "Evidence Detail"
    6. Comprobar que se descarga un pdf con la misma información que había en la página "Evidence Detail"
12. Solicitar precio de mercado:
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón MARKET PRICE
    3. Rellenar el formulario.
    4. Hacer click en el botón SUBMIT
    5. Comprobar que se carga la página de búsqueda de nuevo y se envía un correo con la solicitud.
13. Editar detalle de evidencia:
    1. Iniciar sesión con un usuario administrador
    2. Hacer click en el botón SEARCH
    3. Guardar el valor "File Reference" del primer resultado
    4. Hacer click en el botón del primer resultado
    5. Hacer click en el botón SAVE de la ventana "Evidence Detail"
    6. Hacer click en el botón de la misma evidencia.
    7. Comprobar que los datos se han guardado correctamente.

5.2. Cómo: separando páginas de test

Para facilitar el mantenimiento de la automatización con Selenium se han creado clases base asociadas respectivamente a páginas web (de tipo página) y a pruebas o test (de tipo test). Llamaremos clases base a las que generalizan elementos usados por las otras dos. Las clases del tipo página heredan de una página base y las del tipo test de una clase de tests base:

- Clase *Base.cs*: de esta heredan las clases de tipo página. Contiene los métodos usados en todas las páginas. Por ejemplo, el método que paraliza la ejecución del test hasta que un elemento pasado por parámetro es visible en la página.
- Clase *BaseTest.cs*: de la que heredan las clases de tipo tests. Contienen los métodos que manipulan el controlador de la página. Por ejemplo, la inicialización del controlador (también conocido como *WebDriver*) o abrir y cerrar el navegador. Hemos usado una librería que permite construir un controlador del navegador Chrome.

Se ha utilizado el patrón de diseño *Page Object Model* (POM), muy frecuente en automatización de pruebas. El POM es un patrón de diseño en Selenium que crea un repositorio de objetos para almacenar todos los elementos web: títulos, cajas de texto, botones.... Es útil para reducir la duplicidad de código y mejora la mantenibilidad de los casos de prueba. Consiste en considerar cada página web de una aplicación como una clase de C#

(o el lenguaje elegido), separando estas clases de las clases de tests, como explicaremos en el capítulo 6.

Siguiendo este modelo, por cada página web de la aplicación, se ha creado una clase que la representa. En esta clase se crea un atributo para representar a cada elemento web y se crean métodos para operar sobre dichos elementos web. Para lograr una solución más fácil de entender y mantener se ha nombrado a cada método de la forma más descriptiva posible. Por ejemplo en la página titulada "Detalle de Evidencia" se ha creado un método que representa un click sobre el botón de "Guardar":

```
class EvidenceDetail : Base
  {
    [FindsBy(How = How.Id, Using = "save-action")]
    private IWebElement _saveButton;
   ...
    private void ClickOnSaveButton()
    {
      base.WaitUntilElementIsVisible(_saveButton);
      _saveButton.Click();
    }
}
```

6. Implementación

Se ha dividido la automatización de pruebas en 2 carpetas llamadas **Pages** y **Tests**, como puede verse en el esquema de la figura 6.

6.1. Carpeta "Pages"

Cada una de las clases de esta carpeta corresponde a una página web y, como hemos explicado antes, incluye tantos atributos como elementos a rastrear y tantos métodos como acciones se quieran realizar sobre estos elementos. Estos elementos se incluirán posteriormente en los tests y corresponden a campos de formularios, botones, tablas de resultados, etc.

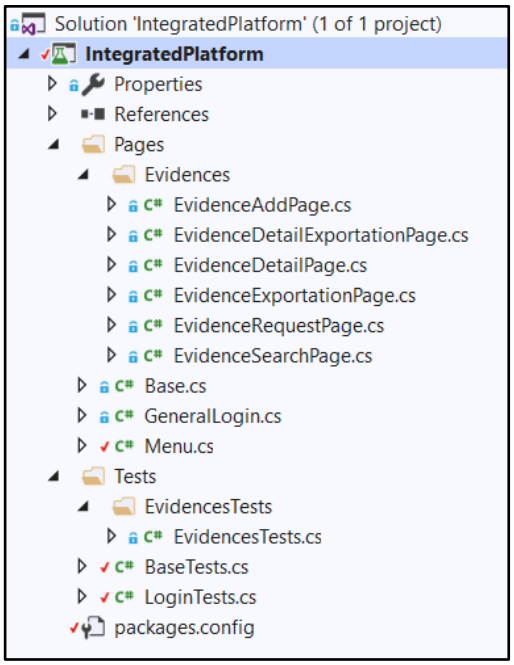

Figura 6. Esquema de solución de la automatización de pruebas

Por ejemplo, el rastreo de los elementos de la página de búsqueda de evidencias, denominada en la Figura 6 *EvidenceSearchPage.cs*, se muestra en la Figura 7. Con la partícula *FindsBy* localizamos cada uno de los filtros de búsqueda y lo guardamos como atributo de tipo *IWebElement*.

En este caso se ha usado el id de cada filtro para localizarlo, pero también se podría localizar por name, CSS selector o por XPath (XML Path Language). Este último método es muy frecuente en la automatización de pruebas. Consiste en una ruta estática o relativa de la posición del elemento en la página html. Los atributos son de tipo *private* para que su contenido no pueda ser alterado con llamadas externas a la clase, desde las clases de tests. Toda consulta o manipulación de estos atributos se hace a través de los métodos de la clase.

```
using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;
using SeleniumExtras.PageObjects;

namespace IntegratedPlatform.Pages
{
    25 references | Urra Amaia, 300 days ago | 1 author, 5 changes
    public class EvidenceSearch : Base
    {

        [FindsBy(How = How.Id, Using = "evidenceTypeField")]
        private IWebElement _evidenceTypeComboBox;

        [FindsBy(How = How.Id, Using = "goodsDescriptionField")]
        private IWebElement _goodsDescription;

        [FindsBy(How = How.Id, Using = "HSCodeField")]
        private IWebElement _hsCode;

        [FindsBy(How = How.Id, Using = "originCountriesField")]
        private IWebElement _countryOfOriginComboBox;
```

Figura 7. Localización de elementos de la página de búsqueda de evidencias por id

Vamos a ver un segundo ejemplo extraído también de la página de búsqueda de evidencias, *EvidenceSearchPage.cs.* En este caso se trata del método para rellenar formularios denominado *FillForm,* que se presenta en la Figura 8, y que recibe el valor de los filtros a buscar y en caso de no completarlos los rellena con un string vacío (“”).

Desde este método se invoca a los métodos *SelectInElement* y *SendTextToElement*. Se trata de dos métodos genéricos creados en la clase *Base.cs* (ver Figura 6) para seleccionar un valor de un cuadro desplegable y para rellenar un campo de tipo texto respectivamente. Para ello, reciben dos parámetros: el atributo *IWebElement* de la página que queremos rellenar (ver Figura 7) y el valor que ha recibido el método *FillForm* en un parámetro. La Figura 9 muestra la implementación de dichos métodos en la clase *Base.cs*.

```
public void FillForm(string evidenceType = "", string description = "", string hsCode = "", string originCountryId = "",
    string unitId = "", string importer = "", string exporter = "", string fileReference = "", string supplyCountryId = "",
    string currencyId = "", string brand = "", string model = "", string sourceId = "", string fromDate = "", string toDate = "")
{
    base.SelectInElement(_evidenceTypeComboBox, "type_" , evidenceType);
    base.SendTextToElement(_goodsDescription, description);
    base.SendTextToElement(_hsCode, hsCode);
    this.SelectInElement(_countryOfOriginComboBox, "countryOrigin_" , originCountryId);
    this.ClickOnAdvanceSearchButton();
    base.SelectInElement(_unitComboBox, "unit_" , unitId);
    base.SendTextToElement(_importer, importer);
    base.SendTextToElement(_importer, exporter);
    base.SendTextToElement(_fileReference, fileReference);
    base.SelectInElement(_countryOfSupplyCombobox, "supplyCountry_" , supplyCountryId);
    base.SelectInElement(_currencyCombobox, "currency_" , currencyId);
    base.SendTextToElement(_brand, brand);
    base.SendTextToElement(_model, model);
    base.SelectByPosition(_sourceCombobox, 1);
    base.SendTextToElement(_fromDate, fromDate);
    base.SendTextToElement(_toDate, toDate);
    this.ClickOnSearchButton();
}
```

Figura 8. Método FillForm de la página EvidenceSearchPage.cs

```
27 references | Urra Amaia, 332 days ago | 1 author, 1 change
protected void SendTextToElement(IWebElement _element, string textToSend)
{
    WaitUntilElementIsVisible(_element);
    _element.Click();
    _element.Clear();
    _element.SendKeys(textToSend);
}

Urra Amaia, 300 days ago | 1 author, 2 changes
protected void SelectInElement(IWebElement _element, string elementBase, string elementId)
{
    if (!string.IsNullOrEmpty(elementId))
    {
        _element.Click();
        IWebElement elementById = driver.FindElement(By.Id(elementBase + elementId));
        this.WaitUntilElementIsVisible(elementById);
        elementById.Click();
    }
}
```

Figura 9. SendTextToElement y SelectInElement de la página BasePage.cs

6.2. Carpeta “Tests”

Los tests se incluyen en una clase de la de tipo Tests identificada con la partícula de Selenium *[TestClass]* y cada test o prueba se identifica con la partícula *[Test]*. Esto permite que queden identificados como casos de prueba y que se puedan ejecutar a través del servicio de Selenium. Los métodos de las clases se llaman desde la Clase *EvidenceTests*.

Por ejemplo, la Figura 10 muestra la implementación del test “Realizar una búsqueda” *(TestSearch)*. En este ejemplo se ha seguido el patrón AAA *(Arrange Act Assert)*:

1. ***Arrange*** (preparar): se preparan las variables y clases que necesitaremos para poder ejecutar la prueba.
2. ***Act*** (actuar): cuerpo de la prueba. Se ejecutan todas las acciones por medio de los métodos de las clases que hemos inicializado en la sección *Arrange*.
3. ***Assert*** (afirmar): se comprueban los resultados de la prueba. Para ello, se ha usado la clase *Assert* proporcionada por Selenium.

El patrón AAA se emplea frecuentemente tanto en pruebas unitarias como automatizadas *END TO END*. También se recomiendan otros patrones como el *Given-When-Then* (Dado que-Cuando-Entonces).

```
using IntegratedPlatform.Pages;
using Assert = Microsoft.VisualStudio.TestTools.UnitTesting.Assert;

namespace IntegratedPlatform.Tests
{
    [TestClass]
    0 references | Urra Amaia, 253 days ago | 2 authors, 4 changes
    public class EvidencesTests : BaseTests
    {
        [Test]
        | 0 references | denissegomezc, 281 days ago | 2 authors, 2 changes
        public void TestSearch()
        {
            //Arrange
            Menu menuEvidences = new Menu(base.webDriver);
            EvidenceSearch Search = new EvidenceSearch(base.webDriver);
            string goodsDescription = "Test 2";
            //Act
            base.LoginAdmin();
            menuEvidences.ClickEvidences();
            Search.FillForm(evidenceType: "TE", importer: "Importer2", description: goodsDescription,
                            unitId: "486", originCountryId: "163");
            //Assert
            Assert.AreEqual(goodsDescription, Search.ObtainGoodsDescription());
        }
    }
}
```

Figura 10. Implementación del test “Realizar una búsqueda” *(TestSearch)*

```
namespace IntegratedPlatform.Tests
{
    1 reference | Urra Amaia, 254 days ago | 3 authors, 9 changes
    public class BaseTests
    {
        protected IWebDriver webDriver;
        protected NgWebDriver ngDriver;

        [SetUp]
        0 references | denissegomezc, 293 days ago | 2 authors, 4 changes
        public void SetUp() {
            webDriver = new ChromeDriver();
            webDriver.Navigate().GoToUrl("https://bcnqaweb10/IntegratedPlatform.WebUI/login");
            webDriver.Manage().Window.Maximize();
        }

        [TearDown]
        0 references | amurra, 336 days ago | 1 author, 1 change
        public void Clear()
        {
            webDriver.Close();
            webDriver.Quit();
        }
    }

}
```

Figura 11. Implementación de la clase BaseTests

La clase de *EvidenceTests* hereda de la clase *BaseTest*, cuyo principal cometido es inicializar una instancia del controlador del explorador al lanzar los tests de la clase *EvidenceTests* y cerrar la sesión del controlador tras finalizar su ejecución.

Las clases de la carpeta *Pages (como por ejemplo: EvidenceDetailPage.cs)*, por convenio, también heredan de la clase *Base* principalmente para contener métodos comunes a todas las páginas. Por ejemplo, el método GoDown de la Figura 12 emula un clic sobre la tecla "fin" de un teclado común.

```
public void GoDown()
{
    this.ScrollDownToBottom();
}
```

Figura 12. Método perteneciente a la clase Base

7. Implantación

Durante el desarrollo, se pretendía incluir este proyecto de automatización en una ejecución automática nocturna en el entorno de Calidad. De esta forma, cada mañana, los QAs hubiéramos podido comprobar si los desarrollos realizados el día anterior habían incluido nuevos defectos en la funcionalidad existente. Esto no se llegó a completar. En este momento, la aplicación ha sido completamente implantada en producción y el entorno de calidad ha dejado de ser operativo.

Las ejecuciones que se han hecho de esta automatización de pruebas se lanzaron a diario de forma manual a lo largo del desarrollo de la aplicación y garantizaron que la aplicación era estable.

Desde que se implantó VQ Light, solo hemos tenido una incidencia en producción derivada de un mal uso por parte de los usuarios de la aplicación.

8. Conclusiones

A lo largo del desarrollo del TFG he identificado algunas lecciones aprendidas que me gustaría destacar. Quizá la principal es *cómo distinguir si conviene o no automatizar un caso de prueba concreto*. No todos los casos de prueba son automatizables pero, aunque lo sean, no siempre merece la pena invertir el esfuerzo que exigen. Es importante tener alguna noción para seleccionar las pruebas a automatizar. En este TFG, al no tener muy claro qué convenía hacer, se han automatizado todos los elementos de todos los formularios. Pero hay que tener presente que la automatización de un caso de prueba es una labor muy pesada, repetitiva y que consume muchísimo tiempo. La reflexión posterior sobre el interés de bastantes casos implementados es que no ha merecido la pena realizar tanto esfuerzo para lograr tan poco. Se trata de garantizar la estabilidad del producto. Así que en futuros proyectos seleccionaré únicamente los elementos más imprescindibles al rellenar un formulario, incluyendo algunos campos que hayan generado defectos recurrentes en el pasado.

La segunda lección es que invertir un esfuerzo inicial en *la búsqueda de librerías y herramientas apropiadas, puede simplificar significativamente la tarea*. He tardado en

identificar una librería que habría facilitado la interacción con el código del *frontend.* La interfaz de la aplicación se estaba desarrollando en Angular, framework desarrollado por Google, y existe una herramienta de testing de software libre, denominada Proactor y creada también por Google, especializada en testear aplicaciones desarrolladas con Angular. Actualmente me estoy formando en dicha herramienta que pondré a prueba en futuros proyectos.

La ejecución de los tests automáticos creados fue mucho más rápida de lo esperado. Incluyendo los tiempos de espera, todas las pruebas se ejecutaban en menos de 30 minutos. Comparando este tiempo con las 15 horas necesarias para la regresión manual, la mejora lograda es sustancial. Desgraciadamente, el desarrollo de este proyecto quedó paralizado por falta de nuevos contratos, por lo que, hasta el momento apenas le hemos podido sacar partido a esta mejora de tiempo. Sin embargo, la experiencia ha merecido la pena para tener un primer contacto con este tipo de soluciones.

A nivel personal, el desarrollo de este proyecto me ha servido para poder iniciarme en la automatización de pruebas con Selenium y gracias a los conocimientos adquiridos, me han asignado el mantenimiento y mejora de las pruebas de otro proyecto de la empresa llamado Customer Portal. En esta aplicación, la estabilidad es más importante aún si cabe, debido a que en este proyecto, tratamos con clientes externos a la empresa.